L-99: Ninety-Nine Lisp Problems

A Complete Reference Guide for Common Lisp Practice

Original author: Werner Hett, Berne University of Applied Sciences (P-99: Ninety-Nine Prolog Problems)

Lisp adaptation: L-99, Institute of Computing, University of Campinas (IC/Unicamp)

Scope: 88 distinct problems numbered P01–P99 (with gaps), organized into 7 thematic sections

This document: Reorganized by Lisp domain with full original text, simplified descriptions, and domain explanations

I	**Abstract** The L-99 problem set is a widely used exercise collection for learning functional programming in Lisp-family languages. Originally written for Prolog by Werner Hett and subsequently adapted for Common Lisp, the problems progress from elementary list manipulation through number theory, logic, tree and graph algorithms, to advanced constraint-satisfaction puzzles. Each problem is annotated with its difficulty (★ easy, ★★ medium, ★★★ hard), classified by the Lisp domain it exercises, and accompanied by a simplified one-line task description for rapid scanning. The full original problem statement and REPL examples are included so this document serves as a complete, self-contained reference without requiring internet access.
II	**Contents** 1. Working with Lists — 28 problems 2. Arithmetic — 11 problems 3. Logic and Codes — 5 problems 4. Binary Trees — 18 problems 5. Multiway Trees — 6 problems 6. Graphs — 10 problems 7. Miscellaneous Problems — 10 problems

III	**Column Legend** **ID** — Original problem number (P01–P99) for cross-referencing. **Diff** — ★ straightforward, ★★ requires thought, ★★★ challenging. **Lisp Domain** — The functional programming concept or data-structure area. **What This Domain Means** — A plain-language explanation of the domain term to aid comprehension. **Simplified Task** — The core idea in minimal words for cognitive efficiency. **Original Problem Text & Example** — Complete original statement with REPL examples.

1. Working with Lists

Fundamental cons-cell operations: traversal via CAR/CDR, recursive construction, filtering, grouping, sorting, data compression, combinatorics, and randomized selection.

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example

P01	★	**List Traversal**	Walking through elements one by one using CAR (first) and CDR (rest)	Return the last element	Find the last box of a list.
					
					`* (my-last '(a b c d))`
					`(D)`

P02	★	**List Traversal**	Walking through elements one by one using CAR (first) and CDR (rest)	Return the second-to-last element	Find the last but one box of a list.
					
					`* (my-but-last '(a b c d))`
					`(C D)`

P03	★	**List Indexing**	Accessing a specific element by its numeric position	Access element at position K	Find the K'th element of a list. The first element in the list is number 1.
					
					`* (element-at '(a b c d e) 3)`
					`C`

P04	★	**List Measurement**	Computing a numeric property of the list (e.g. length)	Count list length	Find the number of elements of a list.
P05	★	**List Transform**	Producing a structurally different list from the input	Build reversed copy	Reverse a list.
P06	★	**List Predicate**	Returning T or NIL based on a property of the list	Check palindrome property	Find out whether a list is a palindrome. A palindrome can be read forward or backward; e.g. (x a m a x).

P07	★★	**Nested Structures**	Lists that contain other lists as elements — recursive depth	Recursively unwrap all sublists	Flatten a nested list structure. Transform a list, possibly holding lists as elements into a flat list by replacing each list with its elements (recursively).
					Hint: Use the predefined functions list and append.
					
					`* (my-flatten '(a (b (c d) e)))`
					`(A B C D E)`

P08	★★	**List Filtering**	Selecting or removing elements based on a condition	Remove adjacent duplicates	Eliminate consecutive duplicates of list elements. If a list contains repeated elements they should be replaced with a single copy of the element. The order of the elements should not be changed.
					
					`* (compress '(a a a a b c c a a d e e e e))`
					`(A B C A D E)`

P09	★★	**List Grouping**	Collecting elements into sub-collections by a criterion	Group adjacent equals into sublists	Pack consecutive duplicates of list elements into sublists. If a list contains repeated elements they should be placed in separate sublists.
					
					`* (pack '(a a a a b c c a a d e e e e))`
					`((A A A A) (B) (C C) (A A) (D) (E E E E))`

P10	★	**Data Compression**	Reducing data size by encoding repeated patterns	Encode as (count element) pairs	Run-length encoding of a list. Use the result of problem P09 to implement the so-called run-length encoding data compression method. Consecutive duplicates of elements are encoded as lists (N E) where N is the number of duplicates of the element E.
					
					`* (encode '(a a a a b c c a a d e e e e))`
					`((4 A) (1 B) (2 C) (2 A) (1 D) (4 E))`

P11	★	**Data Compression**	Reducing data size by encoding repeated patterns	Singletons stay plain	Modified run-length encoding. Modify the result of problem P10 in such a way that if an element has no duplicates it is simply copied into the result list. Only elements with duplicates are transferred as (N E) lists.
					
					`* (encode-modified '(a a a a b c c a a d e e e e))`
					`((4 A) B (2 C) (2 A) D (4 E))`

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example
P12	★★	**Data Compression**	Reducing data size by encoding repeated patterns	Expand encoded list back	Decode a run-length encoded list. Given a run-length code list generated as specified in problem P11. Construct its uncompressed version.

P13	★★	**Data Compression**	Reducing data size by encoding repeated patterns	Encode directly (no P09)	Run-length encoding of a list (direct solution). Implement the so-called run-length encoding data compression method directly. I.e. don't explicitly create the sublists containing the duplicates, as in problem P09, but only count them.
					
					`* (encode-direct '(a a a a b c c a a d e e e e))`
					`((4 A) B (2 C) (2 A) D (4 E))`

P14	★	**List Construction**	Building a new list element by element via CONS	Double every element	Duplicate the elements of a list.
					
					`* (dupli '(a b c c d))`
					`(A A B B C C C C D D)`

P15	★★	**List Construction**	Building a new list element by element via CONS	Repeat each element N times	Replicate the elements of a list a given number of times.
					
					`* (repli '(a b c) 3)`
					`(A A A B B B C C C)`

P16	★★	**List Filtering**	Selecting or removing elements based on a condition	Remove every Nth element	Drop every N'th element from a list.
					
					`* (drop '(a b c d e f g h i k) 3)`
					`(A B D E G H K)`

P17	★	**List Splitting**	Dividing one list into two or more separate lists	Divide at position N	Split a list into two parts; the length of the first part is given. Do not use any predefined functions.
					
					`* (split '(a b c d e f g h i k) 3)`
					`((A B C) (D E F G H I K))`

P18	★★	**List Slicing**	Extracting a contiguous subsequence by index range	Extract elements I..K	Extract a slice from a list. Given two indices, I and K, the slice is the list containing the elements between the I'th and K'th element of the original list (both limits included). Start counting the elements with 1.
					
					`* (slice '(a b c d e f g h i k) 3 7)`
					`(C D E F G)`

P19	★★	**List Rotation**	Circular shifting — elements wrap around the ends	Shift elements cyclically	Rotate a list N places to the left.
					Hint: Use the predefined functions length and append, as well as the result of problem P17.
					
					`* (rotate '(a b c d e f g h) 3)`
					`(D E F G H A B C)`
					`* (rotate '(a b c d e f g h) -2)`
					`(G H A B C D E F)`

P20	★	**List Modification**	Inserting or removing at a specific position	Delete element at position K	Remove the K'th element from a list.
					
					`* (remove-at '(a b c d) 2)`
					`(A C D)`

P21	★	**List Modification**	Inserting or removing at a specific position	Place new element at index K	Insert an element at a given position into a list.
					
					`* (insert-at 'alfa '(a b c d) 2)`
					`(A ALFA B C D)`

P22	★	**List Generation**	Creating a list from a numeric specification (e.g. range)	Generate integer range	Create a list containing all integers within a given range. If first argument is smaller than second, produce a list in decreasing order.
					
					`* (range 4 9)`
					`(4 5 6 7 8 9)`

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example

P23	★★	**Randomness**	Non-deterministic selection using random number generation	Pick N random items	Extract a given number of randomly selected elements from a list. The selected items shall be returned in a list.
					Hint: Use the built-in random number generator and the result of problem P20.
					
					`* (rnd-select '(a b c d e f g h) 3)`
					`(E D A)`

P24	★	**Randomness**	Non-deterministic selection using random number generation	N unique random numbers	Lotto: Draw N different random numbers from the set 1..M. The selected numbers shall be returned in a list.
					Hint: Combine the solutions of problems P22 and P23.
					
					`* (lotto-select 6 49)`
					`(23 1 17 33 21 37)`

P25	★	**Randomness**	Non-deterministic selection using random number generation	Shuffle a list	Generate a random permutation of the elements of a list.
					Hint: Use the solution of problem P23.
					
					`* (rnd-permu '(a b c d e f))`
					`(B A D C E F)`

P26	★★	**Combinatorics**	Systematically enumerating all possible combinations or partitions	All K-element subsets	Generate the combinations of K distinct objects chosen from the N elements of a list. In how many ways can a committee of 3 be chosen from a group of 12 people? We all know that there are C(12,3) = 220 possibilities.
					
					`* (combination 3 '(a b c d e f))`
					`((A B C) (A B D) (A B E) ... )`

P27	★★	**Combinatorics**	Systematically enumerating all possible combinations or partitions	Partition into sized groups	Group the elements of a set into disjoint subsets.
					a) In how many ways can a group of 9 people work in 3 disjoint subgroups of 2, 3 and 4 persons?
					b) Generalize the above function in a way that we can specify a list of group sizes.
					
					`* (group3 '(aldo beat carla david evi flip gary hugo ida))`
					`(((ALDO BEAT) (CARLA DAVID EVI) (FLIP GARY HUGO IDA)) ...)`

P28	★★	**Sorting**	Ordering elements by a comparison function	Sort sublists by length	Sorting a list of lists according to length of sublists.
					a) Sort elements according to their length.
					b) Sort according to their length frequency.
					
					`* (lsort '((a b c) (d e) (f g h) (d e) (i j k l) (m n) (o)))`
					`((O) (D E) (D E) (M N) (A B C) (F G H) (I J K L))`

2. Arithmetic

Number theory in Lisp: primality testing, GCD, Euler's totient function, prime factorization, and Goldbach's conjecture.

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example

P31	★★	**Primality**	Testing whether a number has no divisors other than 1 and itself	Test if integer is prime	Determine whether a given integer number is prime.
					
					`* (is-prime 7)`
					`T`

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example

P32	★★	Number Theory	Properties of and relationships between integers (GCD, coprimality, totient)	Compute GCD (Euclid)	Determine the greatest common divisor of two positive integer numbers. Use Euclid's algorithm.
					
					`* (gcd 36 63)`
					`9`

P33	★	Number Theory	Properties of and relationships between integers (GCD, coprimality, totient)	Check if GCD = 1	Determine whether two positive integer numbers are coprime. Two numbers are coprime if their greatest common divisor equals 1.
					
					`* (coprime 35 64)`
					`T`

P34	★★	Number Theory	Properties of and relationships between integers (GCD, coprimality, totient)	Count coprimes below m	Calculate Euler's totient function phi(m). Euler's so-called totient function phi(m) is defined as the number of positive integers r (1 <= r < m) that are coprime to m. Euler's totient function plays an important role in RSA cryptography.
					
					`* (totient-phi 10)`
					`4`

P35	★★	Factorization	Decomposing a number into its prime building blocks	Flat list of prime factors	Determine the prime factors of a given positive integer. Construct a flat list containing the prime factors in ascending order.
					
					`* (prime-factors 315)`
					`(3 3 5 7)`

P36	★★	Factorization	Decomposing a number into its prime building blocks	Factors with exponents	Determine the prime factors of a given positive integer (2). Construct a list containing the prime factors and their multiplicity.
					Hint: The problem is similar to problem P10.
					
					`* (prime-factors-mult 315)`
					`((3 2) (5 1) (7 1))`

P37	★★	Number Theory	Properties of and relationships between integers (GCD, coprimality, totient)	Efficient phi via factorization	Calculate Euler's totient function phi(m) (improved). If the list of the prime factors of a number m is known in the form of problem P36 then phi(m) can be efficiently calculated: phi(m) = (p1-1)*p1**(m1-1) * (p2-1)*p2**(m2-1) * ...
P38	★	Benchmarking	Measuring and comparing algorithm performance empirically	Compare P34 vs P37	Compare the two methods of calculating Euler's totient function. Take the number of basic operations as a measure for efficiency. Try to calculate phi(10090).
P39	★	Primality	Testing whether a number has no divisors other than 1 and itself	All primes in a range	A list of prime numbers. Given a range of integers by its lower and upper limit, construct a list of all prime numbers in that range.

P40	★★	Conjecture	Exploring unproven mathematical claims computationally	Two primes summing to even N	Goldbach's conjecture. Goldbach's conjecture says that every positive even number greater than 2 is the sum of two prime numbers.
					
					`* (goldbach 28)`
					`(5 23)`

P41	★★	Conjecture	Exploring unproven mathematical claims computationally	Goldbach pairs for a range	A list of Goldbach compositions. Given a range of integers by its lower and upper limit, print a list of all even numbers and their Goldbach composition.
					
					`* (goldbach-list 9 20)`
					`10 = 3 + 7`
					`12 = 5 + 7`
					`14 = 3 + 11`
					`16 = 3 + 13`
					`18 = 5 + 13`
					`20 = 3 + 17`

3. Logic and Codes

Boolean logic evaluation, truth-table generation, and information-theoretic encoding. Gray codes and Huffman coding.

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example

P46	★★	**Boolean Logic**	Evaluating expressions over True/False values (AND, OR, NOT, ...)	Evaluate all input combinations	Truth tables for logical expressions. Define functions and, or, nand, nor, xor, impl and equ (for logical equivalence). Write a function table which prints the truth table of a given logical expression in two variables.
					
					`* (table 'A 'B '(and A (or A B)))`
					`true true true`
					`true nil true`
					`nil true nil`
					`nil nil nil`

P47	★	**Boolean Logic**	Evaluating expressions over True/False values (AND, OR, NOT, ...)	Parse infix boolean expressions	Truth tables for logical expressions (2). Modify problem P46 by accepting expressions written in infix notation, with all parentheses present.
					
					`* (table 'A 'B '(A and (A or (not B))))`
					`true true true`
					`true nil true`
					`nil true nil`
					`nil nil nil`

P48	★★	**Boolean Logic**	Evaluating expressions over True/False values (AND, OR, NOT, ...)	Generalize to N variables	Truth tables for logical expressions (3). Generalize problem P47 in such a way that the logical expression may contain any number of logical variables.
					
					`* (table '(A B C) '((A and (B or C)) equ`
					`((A and B) or (A and C))))`
					`true true true true ...`

P49	★★	**Encoding**	Mapping data into a compact or structured representation (Gray, Huffman)	Generate N-bit Gray sequence	Gray code. An n-bit Gray code is a sequence of n-bit strings constructed according to certain rules. Find out the construction rules.
					Can you apply the method of result caching to make the function more efficient?
					
					`n=1: ("0" "1")`
					`n=2: ("00" "01" "11" "10")`
					`n=3: ("000" "001" "011" "010" "110" "111" "101" "100")`

P50	★★ ★	**Encoding**	Mapping data into a compact or structured representation (Gray, Huffman)	Build Huffman prefix codes	Huffman code. We suppose a set of symbols with their frequencies, given as a list of (S F) elements. Construct a list of (S C) elements, where C is the Huffman code word for symbol S.
					
					`* (huffman '((a 45) (b 13) (c 12) (d 16) (e 9) (f 5)))`
					`((A "0") (B "101") (C "100") (D "111") (E "1101") (F "1100"))`

4. Binary Trees

Binary trees represented as (X L R) or NIL. Construction, validation, traversal, balancing, layout algorithms for visualization, and serialization.

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example

P54A	★	**Tree Predicate**	Testing whether a structure satisfies tree-specific properties	Validate (X L R) structure	Check whether a given expression represents a binary tree. Write a function istree which returns true if and only if its argument is a list representing a binary tree.
					
					`* (istree '(a (b nil nil) nil))`
					`T`
					`* (istree '(a (b nil nil)))`
					`NIL`

P55	★★	**Tree Construction**	Building tree structures from specifications or constraints	All balanced trees for N nodes	Construct completely balanced binary trees. The number of nodes in its left subtree and the number of nodes in its right subtree are almost equal (differ by at most one).
					
					`* (cbal-tree 4)`
					`((X (X NIL NIL) (X NIL (X NIL NIL))) ...)`

P56	★★	**Tree Predicate**	Testing whether a structure satisfies tree-specific properties	Check mirror symmetry	Symmetric binary trees. A binary tree is symmetric if the right subtree is the mirror image of the left subtree.

P57	★★	**Tree Construction**	Building tree structures from specifications or constraints	Build BST from integer list	Binary search trees (dictionaries). Write a function to construct a binary search tree from a list of integer numbers.
					
					`* (construct '(3 2 5 7 1))`
					`(3 (2 (1 nil nil) nil) (5 nil (7 nil nil)))`

P58	★★	**Generate-and-Test**	Create all candidates, then filter by a predicate	Symmetric balanced trees	Apply the generate-and-test paradigm to construct all symmetric, completely balanced binary trees with a given number of nodes. How many such trees are there with 57 nodes?
P59	★★	**Tree Construction**	Building tree structures from specifications or constraints	All AVL-like trees for height H	Construct height-balanced binary trees. The height of left and right subtrees differ by at most one. Generate all solutions for a given height.
P60	★★	**Tree Construction**	Building tree structures from specifications or constraints	Height-balanced trees with N nodes	Construct height-balanced binary trees with a given number of nodes. What is the minimum number of nodes for height H? What is the maximum height with N nodes?
P61	★	**Tree Traversal**	Visiting tree nodes in a defined order (pre/in/post/level)	Count leaf nodes	Count the leaves of a binary tree. A leaf is a node with no successors.
P61A	★	**Tree Traversal**	Visiting tree nodes in a defined order (pre/in/post/level)	Collect leaf values	Collect the leaves of a binary tree in a list.
P62	★	**Tree Traversal**	Visiting tree nodes in a defined order (pre/in/post/level)	Collect internal node values	Collect the internal nodes of a binary tree in a list. An internal node has either one or two non-empty successors.
P62B	★	**Tree Traversal**	Visiting tree nodes in a defined order (pre/in/post/level)	All nodes at depth N	Collect the nodes at a given level in a list. The root node is at level 1.
P63	★★	**Tree Construction**	Building tree structures from specifications or constraints	Build left-adjusted full tree	Construct a complete binary tree. Levels 1..H-1 contain the maximum number of nodes. In level H, all nodes are left-adjusted.
P64	★★	**Tree Layout**	Assigning (x,y) coordinates to nodes for visual rendering	x = inorder pos, y = depth	Layout a binary tree (1). x(v) = position in inorder sequence; y(v) = depth of v in the tree.
P65	★★	**Tree Layout**	Assigning (x,y) coordinates to nodes for visual rendering	Constant spacing per level	Layout a binary tree (2). On a given level, the horizontal distance between neighboring nodes is constant.
P66	★★ ★	**Tree Layout**	Assigning (x,y) coordinates to nodes for visual rendering	Compact symmetric packing	Layout a binary tree (3). Very compact layout while maintaining symmetry. How tight can you pack two subtrees?
P67	★★	**Serialization**	Converting between in-memory tree structures and string representations	Tree <-> string conversion	A string representation of binary trees: a(b(d,e),c(,f(g,))). Write functions for both directions. `a(b(d,e),c(,f(g,)))`

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example
P68	★★	**Serialization**	Converting between in-memory tree structures and string representations	Preorder / inorder sequences	Preorder and inorder sequences of binary trees. a) Write functions preorder and inorder. b) Write the inverse of preorder. c) Given both preorder and inorder, reconstruct the tree.
P69	★★	**Serialization**	Converting between in-memory tree structures and string representations	Dotstring: preorder + dots for NIL	Dotstring representation. Dots (.) are inserted where an empty subtree (nil) is encountered during preorder traversal. `"ABD..E..C.FG..."`

5. Multiway Trees

Trees with arbitrary branching. Represented as (X C1 C2 ... CN) or a single symbol for leaves.

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example

P70B	★	**Tree Predicate**	Testing whether a structure satisfies tree-specific properties	Validate multiway tree structure	Check whether a given expression represents a multiway tree.
					
					`* (istree '(a (f g) c (b d e)))`
					`T`

P70C	★	**Tree Measurement**	Computing numeric properties (path length, height, node count)	Count all nodes	Count the nodes of a multiway tree.
P70	★★	**Serialization**	Converting between in-memory tree structures and string representations	Parse node-string into tree	Tree construction from a node string. A special character ^ is inserted on backtrack to previous level. `"afg^^c^bd^e^^^"`
P71	★	**Tree Measurement**	Computing numeric properties (path length, height, node count)	Sum of root-to-node distances	Determine the internal path length: total sum of all path lengths from the root to all nodes.
P72	★	**Tree Traversal**	Visiting tree nodes in a defined order (pre/in/post/level)	Post-order node sequence	Construct the bottom-up order sequence of the tree nodes.
P73	★★	**Serialization**	Converting between in-memory tree structures and string representations	Convert tree <-> S-expression	Lisp-like tree representation. A node with successors is always the first element in a list, followed by its children.

6. Graphs

Graph representations, pathfinding, spanning trees, isomorphism, coloring, DFS, connected components, and bipartiteness.

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example
P80	★★ ★	**Representation**	Converting between equivalent data forms (adjacency list, edge list, etc.)	Convert between graph forms	Conversions. Write functions to convert between adjacency-list form, edge-list (graph-term) form, and human-friendly form.
P81	★★	**Pathfinding**	Finding routes through connected nodes in a graph	All acyclic paths A -> B	Path from one node to another one. Find all acyclic paths between two given nodes.

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example
P82	★	**Cycle Detection**	Finding closed loops — paths that return to the start	Find cycle starting at node	Cycle from a given node. Find a closed path (cycle) starting at a given node.
P83	★★	**Spanning Trees**	Finding minimal connected subgraphs covering all nodes	Generate all spanning trees	Construct all spanning trees of a given graph.
P84	★★	**Spanning Trees**	Finding minimal connected subgraphs covering all nodes	Find minimum spanning tree	Construct the minimal spanning tree of a given labeled (weighted) graph.
P85	★★	**Graph Isomorphism**	Checking if two graphs have identical structure under relabeling	Check structural equivalence	Graph isomorphism. Determine if there is a bijection between node sets that preserves adjacency.
P86	★★	**Graph Coloring**	Assigning colors to nodes so no two adjacent nodes share a color	Color with min colors	Node degree and graph coloration. Use Welsh-Powell's algorithm (decreasing degree order).
P87	★★	**Graph Traversal**	Systematically visiting all reachable nodes (DFS, BFS)	DFS node visit order	Depth-first order graph traversal starting from a given node.
P88	★★	**Graph Components**	Identifying disconnected subgraphs within a larger graph	Find connected subgraphs	Connected components. Split a graph into its connected components.
P89	★★	**Graph Predicate**	Testing global graph properties (bipartiteness, planarity, ...)	Check 2-colorability	Bipartite graphs. Determine whether a given graph is bipartite.

7. Miscellaneous Problems

Classic CS puzzles: backtracking, constraint satisfaction, puzzle solving, string processing, and parsing.

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example
P90	★★	**Backtracking**	Systematic trial-and-error: try a choice, undo if it fails	Place 8 non-attacking queens	Eight queens problem. Place eight queens on a chessboard so that no two queens attack each other.
P91	★★	**Backtracking**	Systematic trial-and-error: try a choice, undo if it fails	Visit every square once	Knight's tour. Find a path for a knight that visits every square of an NxN board exactly once.
P92	★★★	**Constraint Solving**	Finding values that satisfy a set of simultaneous rules	Label tree nodes uniquely	Von Koch's conjecture. Number nodes 1..N so that for each edge K, the difference of its node numbers equals K.

P93	★★★	**Constraint Solving**	Finding values that satisfy a set of simultaneous rules	Insert operators to form equation	An arithmetic puzzle. Given a list of integers, insert +,-,*,/ to form a correct equation.
					
					`With (2 3 5 7 11):`
					`2 - 3 + 5 + 7 = 11`
					`2 = (3*5+7)/11`

P94	★★★	**Graph Generation**	Constructing all graphs that satisfy given structural constraints	All K-regular simple graphs	Generate K-regular simple graphs with N nodes. Every node has exactly K edges.

P95	★★	**String Processing**	Transforming between numeric and text representations	Integer to English words	English number words. Write a function fullWords to return integers in full words.
					
					`* (full-words 175)`
					`"one-hundred-seventy-five"`

ID	Diff	Lisp Domain	What This Domain Means	Simplified Task	Original Problem Text & Example
P96	★★	**Parsing**	Analyzing a string's structure according to a formal grammar	Validate against syntax diagram	Syntax checker (Ada identifiers). Check whether a given string is a legal Ada identifier.
P97	★★	**Puzzle**	Solving structured placement or logic problems	Solve 9x9 Sudoku	Sudoku. Write a function to solve a given Sudoku puzzle.
P98	★★ ★	**Puzzle**	Solving structured placement or logic problems	Solve nonogram from clues	Nonograms. Given clue numbers for rows and columns, determine which cells to fill.
P99	★★ ★	**Puzzle**	Solving structured placement or logic problems	Fill crossword grid	Crossword puzzle. Given a framework and a set of words, place the words into the framework.

Sources and Further Reading

• L-99 Lisp Problems — ic.unicamp.br/~meidanis/courses/problemas-lisp/L-99_Ninety-Nine_Lisp_Problems.html

• P-99 Prolog Problems (original) — sites.google.com/site/prologsite/prolog-problems/

• H-99 Haskell Problems — wiki.haskell.org/H-99:_Ninety-Nine_Haskell_Problems

• OCaml Exercises — ocaml.org/exercises

• CL Solutions (informatimago) — informatimago.free.fr/i/develop/lisp/l99/index.html

• CL Solutions (GitHub) — github.com/meagtan/99-lisp-problems

Compiled for personal use as a self-contained practice guide. Original problem texts by Werner Hett. Lisp adaptations from IC/Unicamp. Domain classifications, explanations, and simplified descriptions added for cognitive efficiency.